Universidade Federal de Pernambuco

Centro de informática

Bacharelado em Sistemas de Informação

GenCloud - Uma abordagem para disponibilização de sistemas web baseado em contêineres em um ambiente de nuvem híbrida

Aluno: João Victor Uchôa Vaz (jvuv@cin.ufpe.br)

Orientador: Vinicius Cardoso Garcia (vcg@cin.ufpe.br)

Recife, 2017

Universidade Federal de Pernambuco
Centro de informática
Bacharelado em Sistemas de Informação

GenCloud - Uma abordagem para disponibilização de sistemas web baseado em contêineres em um ambiente de nuvem híbrida

Trabalho de conclusão de curso, apresentado à coordenação de graduação do Centro de Informática da Universidade Federal de Pernambuco, como requisito obrigatório para obtenção do grau de Bacharel no Curso de Sistemas de Informação.

Aluno: João Victor Uchôa Vaz (jvuv@cin.ufpe.br)
Orientador: Vinicius Cardoso Garcia (vcg@cin.ufpe.br)

Recife, 2017

Universidade Federal de Pernambuco
Centro de informática
Bacharelado em Sistemas de Informação

GenCloud - Uma abordagem para disponibilização de sistemas web baseado em contêineres em um ambiente de nuvem híbrida

Trabalho de conclusão de curso, apresentado à coordenação de graduação do Centro de Informática da Universidade Federal de Pernambuco, como requisito obrigatório para obtenção do grau de Bacharel no Curso de Sistemas de Informação.

Trabalho aprovado em: 18 de Julho de 2017.

Vinicius Cardoso Garcia

Recife, 2017

“Aquele que deseja paz e harmonia
nas relações humanas deve sempre lutar
contra o estatismo.”

(Ludwig von Mises)

Agradecimentos

Gostaria de agradecer aos meus pais João Evangelista Vaz e Maria do Socorro Uchôa Vaz por me oferecer todo o suporte moral e econômico para realizar este curso de graduação.

À minha namorada/noiva Wadlla Rocha pela paciência tida durante todo o tempo que precisei para elaborar este trabalho.

Ao professor Vinicius Cardoso Garcia e ao diretor de TI da Genomika Marcel Caraciolo por me orientar neste projeto.

E por fim a todos os meus colegas e professores, que tornaram os anos de graduação engrandecedores e agradáveis.

Resumo

Conforme atividades em equipes de desenvolvimento de software vão ficando mais complexas, problemas acontecem entre diferentes times com diferentes funções [9], acarretando uma perda de desempenho na produção de software. Um dos grandes problemas se encontra entre os membros do desenvolvimento e de operações [1]. A metodologia DevOps propões mudanças no processo interno das equipes de TI a fim de manter uma melhor relação entre esses times com distintas funções [1], porém com papéis importantes na entrega de produtos de software. Para melhor implementa-se esta metodologia, o uso de contêineres de software, especialmente da plataforma Docker [14], se tornaram alternativas bastante utilizadas. Estas tecnologias permitem a criação de ambientes isolados, eliminando os possíveis conflitos entre aplicações, além de possuírem outras funcionalidades que facilitam a gerência de aplicações em uma diversa gama de infraestruturas computacionais [15]. Outros softwares de auxílio e extensão ao uso das ferramentas relatadas anteriormente também surgiram, tornando o objetivo deste trabalho de integrar os serviços de três destes softwares (*Terraform*, *Rancher* e Cattle) com finalidade de tornar o *deploy* de sistemas web mais eficiente, além de permitir que os desenvolvedores possam desenvolver no mesmo ambiente de produção e realizar atividades de construção de infraestrutura, antes do time de operações, de uma maneira mais simplificada.

Palavras-chave: Contêiner, Docker, DevOps, Rancher, Terraform, Cattle, Desenvolvimento, Sistemas Web

Abstract

As activities in software development teams become more complex, problems happen between different teams with different functions [9], leading to a loss of performance in software production. One of the major problems lies between the members of development and operations [1]. The DevOps methodology proposes changes in the internal process in IT teams in order to maintain a better relationship between these teams of different functions [1], but with important roles in the delivery of software products. To better implement this methodology, the use of software containers, especially the Docker platform [14], have become widely used alternatives. These technologies allow the creation of isolated environments, eliminating the possible conflicts between applications, besides having other functionalities that facilitate the management of applications in a diverse range of computational infrastructures [15]. Other support and extension software for the use of previously reported tools have also emerged, making the objective of this work to integrate the services of three of these software (Terraform, Rancher and Cattle) in order to make web systems deploy more efficient, besides allowing that developers can develop in the same production environment and perform infrastructure construction activities, before the operations team, in a more simplified way.

Keywords: Container, Docker, DevOps, Rancher, Terraform, Cattle, Development, Web Systems

Índice de Figuras

Índice de Tabelas

Sumário

1. Introdução

Plataformas de contêineres de software estão ficando cada vez mais populares entre os desenvolvedores de software, destacando-se o Docker [45]. Contêineres proveem isolamento da aplicação, dependências e recursos de maneira similar a uma máquina virtual, mas de maneira mais leve por compartilhar o kernel com o sistema hospedeiro [15].

Paralelamente, a metodologia de desenvolvimento de software DevOps surge com o objetivo de melhorar o relacionamento entre equipes de desenvolvimento e de operações [38]. Estas equipes, muitas vezes, buscam objetivos diferentes. De um lado se tem a equipe de desenvolvimento, que faz uso de metodologias ágeis e precisa lançar funcionalidades em uma frequência alta, e do outro lado, se tem a equipe de operações, que quer manter a aplicação estável [38]. Uma das premissas do DevOps é que o desenvolvedor pode fazer o deploy da aplicação independentemente, e o uso de contêineres, que padronizam o ambiente de execução da aplicação, facilita essa situação [46].

Devido ao crescente uso do Docker com o intuito de suportar os processos da metodologia DevOps, empresas desenvolveram softwares de extensão a esta ferramenta. Este trabalho tem o objetivo de agrupar alguns destes softwares para oferecer uma solução alternativa aos modelos de *deploys* atuais. Estas ferramentas são:

- *Terraform,* (provisionador de infraestrutura)
- *Rancher* (gerenciador de configurações).
- *Cattle* (orquestrador de contêineres docker).

No Capítulo 2 é abordada a fundamentação teórica dos conceitos de DevOps, e das ferramentas de contêineres e Docker, assim como a justificativa do uso destes softwares relacionados a esta metodologia.

O Capítulo 3 traz uma descrição mais precisa das ferramentas que serão utilizadas neste trabalho dentro do contexto apresentado no capítulo anterior e a

justificativa da integração delas para *deploys* de sistemas web, apresentando o projeto GenCloud como um exemplo.

A seguir, no Capítulo 4, há o objetivo de mostrar o que é o projeto GenCloud, compará-lo a um modelo mais tradicional de *deploy* e expor as métricas de implementação e execução destes dois.

Por fim, no Capítulo 5, são apresentadas as considerações finais e as dificuldades encontradas, além de sugestões de trabalhos futuros.

2. Fundamentação e Justificativa

Para continuarem competitivas no mercado, empresas que fornecem serviços de TI precisam encontrar maneiras para entregar seus serviços em um ritmo cada vez mais intenso e com entregas contínuas. Esses serviços são criados e entregues aos clientes por meio de um conjunto de atividades. Essas atividades, por sua vez, são divididas pelos departamentos de negócios, que formulam os requisitos do negócio e dos diferentes departamentos de TI, de desenvolvimento e de operações [1]. Várias metodologias surgiram a fim de tornar esse processo de entrega mais eficiente, tais como a *Ágil* e a *DevOps* [1].

Este trabalho está diretamente ligado ao DevOps, porém, antes de fundamentar sobre esta metodologia, é necessário entender melhor de sua antecessora, a metodologia Ágil.

2.1 Metodologia Ágil

A metodologia *Ágil* consiste de princípios para desenvolvimento de software com quatro valores chaves: indivíduos e interações, software funcional, colaboração do cliente e resposta à mudanças [2].

As práticas observadas no uso de metodologias ágeis de desenvolvimento de software têm as características de acomodar mudanças nos requisitos em qualquer fase do processo de desenvolvimento [3], com seu foco nas fases de negócio e desenvolvimento. Todo o processo de análise de requisitos e desenvolvimento é realizado de modo interativo com o cliente, deixando-o mais satisfeito. A metodologia Ágil, geralmente, quando bem aplicada, resulta em melhores níveis de eficiência, com uma melhor condução no desenvolvimento do produto ou serviço.

Os processos relacionados com a metodologia Ágil mais utilizados são Scrum e eXtreme Programming [4]. O Scrum se concentra em um projeto iterativo na equipe de desenvolvimento auto-organizada, trabalhando em sprints curtos [5]. Ele pressupõe que há requisitos são imprevisíveis e, portanto, o desenvolvimento deve ser capaz de se adaptar às mudanças em ciclos curtos [6]. Essas mudanças permitem que o produto possa ser ajustado o mais rápido possível de acordo com as

necessidades do cliente e a velocidade de entrega seja aumentada [2]. O eXtreme Programming (XP) é um processo que se preocupa como o desenvolvimento do produto ou serviço pode ser mais eficiente, ao invés de focar na gerência do projeto. Ele consiste em práticas e técnicas, como a revisão constante do código, programação em par entre outras [7]. A junção das práticas do Scrum e da eXtreme Programming mostra que elas podem se complementar na projeção e desenvolvimento de projeto [8].

Por mais que esta metodologia tenha trazido benefícios para o desenvolvimento de software, a mesma ainda possui lacunas que afetam a produtividade e que serão abordadas no próximo tópico com a introdução à metodologia DevOps.

2.2 Metodologia DevOps

A metodologia Ágil promove um aumento na eficiência no processo de entrega, porém, quando se olha para o processo completo de desenvolvimento de software, há áreas em que não são cobertas, como por exemplo, a de operações.

O time de operações é composto em parte pelos *sysadmins*, que têm a missão de manter os sistemas funcionando, realizar os *deploys* e *rollbacks* das aplicações, além de ter a responsabilidade de manter o ambiente de produção intacto e avaliar e propor melhorias de forma a manter as aplicações [1].

O time de operações precisa cooperar com o resto da equipe de desenvolvimento. Caso haja falha no processo de cooperação, consequências aparecem, como por exemplo, produtividade mais baixa nas fases de operação e desenvolvimento e menor qualidade no software e serviços para os usuários [9].

Na relação entre a área de desenvolvimento e a de operações pode-se observar seis problemas: “ (1) operações de TI que não estejam envolvidas na especificação de requisitos; (2) má comunicação e fluxo de informação; (3) ambientes de teste insatisfatórios; (4) falta de transferência de conhecimento; (5) sistemas sendo colocados em produção antes de estarem completos; e (6) rotinas operacionais não estabelecidas antes da implantação" [9, p. 394]. O conceito DevOps surge em meio a estes problemas apontados, ele pode ser visto como uma extensão

à metodologia Ágil que promete redução de custos, alto rendimento de inovações, rapidez ao mercado e a melhoria da qualidade [2].

A área de atuação das metodologias Ágil e DevOps em relação aos setores de desenvolvimento e de operações é mostrada na Figura 1. Percebe-se que a Ágil apenas está presente nos processos dos desenvolvimento de software, porém o produto (software) precisa ser rodado nos servidores. Essa operação é realizada pelo time de operações, porém os mesmos precisam se preocupar com a estabilidade do software para não haver problemas de instabilidade para os clientes. Neste momento, o fluxo depara-se com os problemas relatados anteriormente, pois a metodologia Ágil emprega velocidades de produção maiores do que a parte operacional pode suportar.

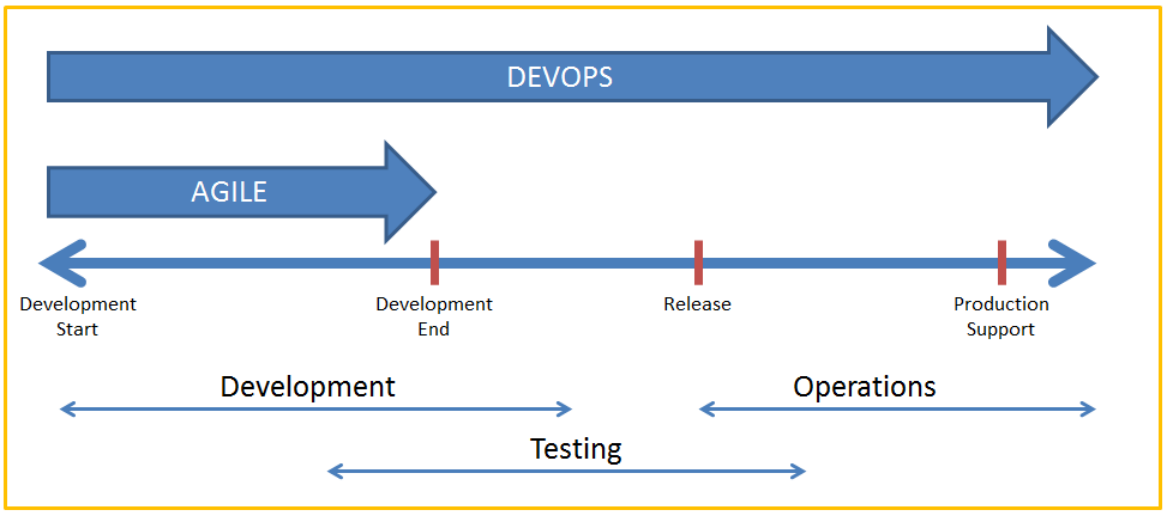

Figura 1 – Comparação entre as metodologias Ágil e DevOps [39].

A metodologia DevOps é um movimento cultural com um conjunto de práticas que ajudam a melhorar a flexibilidade e a efetividade dos processos de negócio, atingindo diferentes departamentos de TI e aplicando uma melhor colaboração entre eles. Ela existe a partir da “Comunicação aberta, incentivo e alinhamento de responsabilidade, respeito e confiança” [10] (p.15). A implementação da cultura DevOps leva a cultura da TI da empresa a ir para um nível mais elevado, levando o foco do movimento a três aspectos: pessoas, processos e ferramentas [2].

Para que os resultados sejam alcançados com uso do DevOps, as organizações precisam aplicar os princípios desta cultura. Estes princípios são apontados como pontos chave para que haja uma melhor eficiência no processo de negócio. O princípio de *systems thinking,* ou *pensamento sistêmico*, foca na

performance do sistema completo, que traz valor ao negócio, ao invés de ter performance em apenas um departamento [11]. Um outro princípio amplifica os *loops* dos *feedbacks* em todo o processo, fazendo com que as correções necessárias sejam continuamente feitas em resposta ao cliente [11]. Como um pilar fundamental para a cultura DevOps, é fundamental considerar o princípio do aspecto da cultura da experimentação contínua que traz consigo o enfrentamento de riscos e o aprendizado com as falhas diante do entendimento da repetição e da prática [11].

2.2.1 Práticas do Devops

Devops é visto como uma extensão da metodologia Ágil [1], portanto muitas práticas e métodos de trabalho definidas no movimento Ágil também farão parte da metodologia DevOps.

Scrum é a prática mais comum no movimento Ágil [1], também sendo aplicada no DevOps com as características de: "foco no cliente, times compostos por pessoas com várias funcionalidades, constantes *feedbacks*, pequenas iterações e a importância de trabalhar no produto" [1, p.33]. Na Tabela 1 mostra-se uma visão geral da metodologia DevOps, com alguns aspectos combinados da metodologia tradicional e Ágil. Um exemplo para isto é quando considera-se o DevOps pela perspectiva Ágil, em que o principal valor Ágil "Indivíduos e interações sobre processos e ferramentas" [2] deve ser levado em consideração.

Categoria	Tradicional	Ágil	Devops
Premissas Fundamentais	Os sistemas estão completamente especificados. Previsíveis, Podem ser construídos através de um meticuloso e extensivo planejamento	Alta qualidade. Adaptativo. Software pode ser desenvolvido por pequenas equipes usando os princípios de melhorias na concepção e baseados em testes rápidos com Feedback à	Alta qualidade. Adaptativo. Serviços podem ser fornecidos por equipes multidisciplinares de ponta a ponta que utilizam os princípios da melhoria contínua, integração, entrega e

		mudanças	possíveis *deploys*
Atribuição de funções	Especialização individual	Equipes auto-organizadas - incentiva o papel da permutabilidade	Equipes multidisciplinares com responsabilidade de ponta a ponta
Comunicação	Formal	Informal	Informal
Regra do Cliente	Importante	Crítica	Crítica
Ciclo do Projeto	Guiada por tarefas ou atividades	Guiado por características do produto	Guiada pelo Valor do negócio de entrega de ponta a ponta
Modelo de Desenvolvimento	Modelo de ciclo de vida (Cachoeira, espiral, ou algum variação)	Modelo de entrega evolucionário	Modelo de Entrega Contínua
Estrutura Organizacional Desejada	Mecanizada (Burocrática com alta formalização)	Orgânica (flexível e ação social cooperativa de participação encorajadora)	Aprendizagem (Interativa, flexível e Pessoas e equipes colaborativas)
Tecnologia	Sem restrição	Tecnologia orientada a objetos	Automação com computação em

			nuvem, infraestrutura como serviço

Tabela 1 - Comparativo entre metodologias de desenvolvimento de software [1]

Na cultura DevOps, a automação de processos é uma parte fundamental para se obter maturidade na equipe de TI. Importantes processos são os de *Continuous Deployment*, ou *Implantação Contínua, Continuous Delivery*, ou *Entrega Contínua* e a Continuous Integration, ou *Integração Contínua*.

A ideia destes processos são a de abordar a automação da publicação de um determinado código em algum tipo de ambiente, tendo passado por testes predeterminados. Esses testes são responsáveis por validar se um novo código atrelado ao antigo está bem integrado (Teste de Integração ou Unitários) e como o código se comporta com o usuário final (Testes de Entrega).

Na *Implantação Contínua*, a *Integração Contínua* (Continuous Integration) e a *Entrega Contínua* (Continuous Delivery) são processos que fazem parte do ciclo da automação do *deploy*. A *Integração Contínua* é responsável por automatizar novos códigos ao existente, já a *Entrega Contínua* é um conjunto de práticas que tem como objetivo garantir que o novo código possa ser implantado no ambiente de produção a qualquer momento, porém entende-se que esta entrega inicial será para um grupo de usuários para validação e após isso, realizar o *deploy* para produção de forma manual. Para a *Implantação Contínua*, precisa-se ter os processos de *Entrega Contínua* e de *Integração Contínua* funcionando como parte da sua rotina [12]. A ideia é entregar valor ao negócio o mais rápido possível e não acumular código. A Figura 2 mostra o limite e o fluxo de cada um dos processos relatados anteriormente.

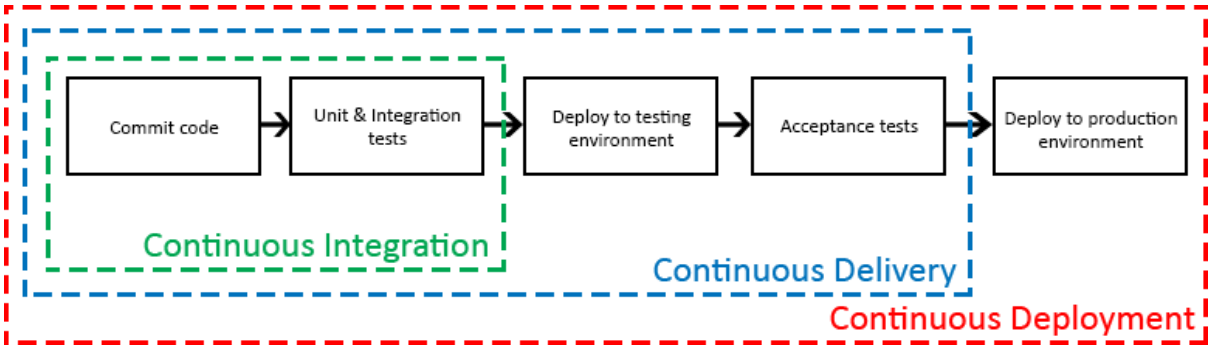

Figura 2 - Comparativo entre a *Integração Contínua*, *Entrega Contínua* e *Implantação Contínua* [35]

Com as práticas do DevOps instauradas na equipe de TI, é permitido o lançamento de uma atualização da aplicação pela equipe desenvolvedora, pois esta deve ter conhecimento de operação suficiente e serviços de automação deste processo para a execução[38].

Para se atingir melhores performances, é essencial que os serviços precisem estar em execução, dar conta da carga de processamento e se recuperar em caso de problemas [13]. Esses serviços são oferecidos com o uso dos *contêineres.*

2.3 Contêiner

Os contêineres representam um método moderno de virtualização de aplicações [14], nos quais garantem um isolamento de outros processos computacionais, além de prover a portabilidade das aplicações [15]. Esta tecnologia, diferente das máquinas virtuais, não virtualiza um sistema operacional por completo, apenas fazendo uso dos recursos do sistema operacional hospedeiro, como por exemplo, o *kernel Linux*, os binários e as bibliotecas. Por serem compartilhados, apenas têm permissão de leitura, deixando-os bastante leves [16].

A Figura 3 apresenta a diferença entre as tecnologias de virtualização com o uso de *hipervisor* e de contêineres. Percebe-se que na virtualização com *hipervisor* existe uma camada que fornece os recursos para a virtualização de outros sistemas operacionais, e na virtualização com contêiner os recursos são compartilhados com o sistema operacional principal e fornecidos para os ambientes virtualizados.

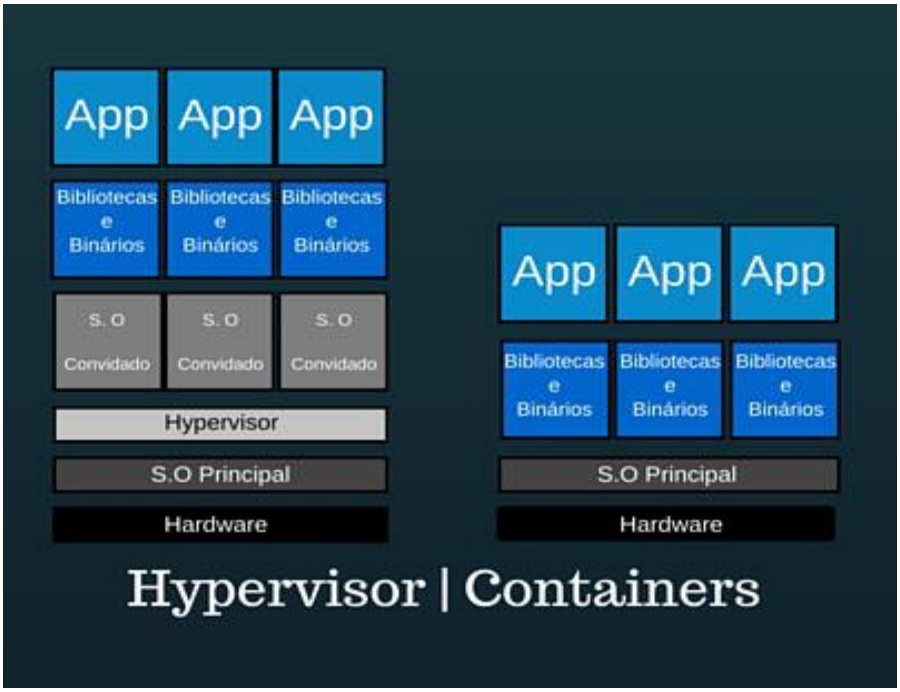

Figura 3 – Comparativo entre máquinas virtuais e contêineres [16]

Para que um contêiner possa garantir seus princípios de gestão, isolamento e segurança, ele precisa da ajuda de alguns recursos do *kernel Linux*, como o *Namespaces* , *cgroups* e *chroot*. Enquanto o *namespaces* faz com que os processos tenham seu próprio ambiente, o *chroot* os isola do resto do ambiente e o *cggroups* tem a função de isolar a utilização de recursos como CPU, memória, I/O de disco, e da rede, a partir de um conjunto de processos [14].

2.3.1 Docker

Os contêineres são ferramentas que trouxeram ganhos para o DevOps, porém, eles são melhores utilizados quando há o apoio de softwares que os gerenciam, como é o caso do *Docker.* Devido a sua facilidade de interação com o humano, essa ferramenta tornou mais fácil a criação, o *deploy* e a execução de aplicações [17]

Apesar do *Docker* ser uma das ferramentas mais conhecidas e usadas no meio dos contêineres, o sistema operacional *Linux* também possui sua própria ferramenta de gerenciamento dos *contêineres* desde 2008*, o LXC* [15]. O *LXC* fornece ao contêiner os recursos necessários de máquina, como memória, processador, isolamento com a máquina hospedeira, bibliotecas de sistema e o *kernel* do *Linux.* O

Docker fez inicialmente uso da ferramenta *LXC*, provendo sua *API* [18] para comunicação deste com o usuário, trazendo uma maior facilidade e agilidade na criação de contêineres, contudo, a substituiu pelo *libcontainer*, ferramenta criada pela própria desenvolvedora.

Atualmente, existem duas entidades que compõem a ferramenta Docker, o *Docker Client* e o *Docker Daemon.* O *Docker Client* é um utilitário que faz com que o humano rode comandos de fácil entendimento, abstraindo comandos complexos [19]. Já o *Docker Daemon* é a camada que se comunica com o *kernel* e faz as chamadas de sistema para criar, operar e gerenciar contêineres. Na Figura 4 mostra-se um desenho da comunicação entre o *Docker Client* e o *Docker Daemon*.

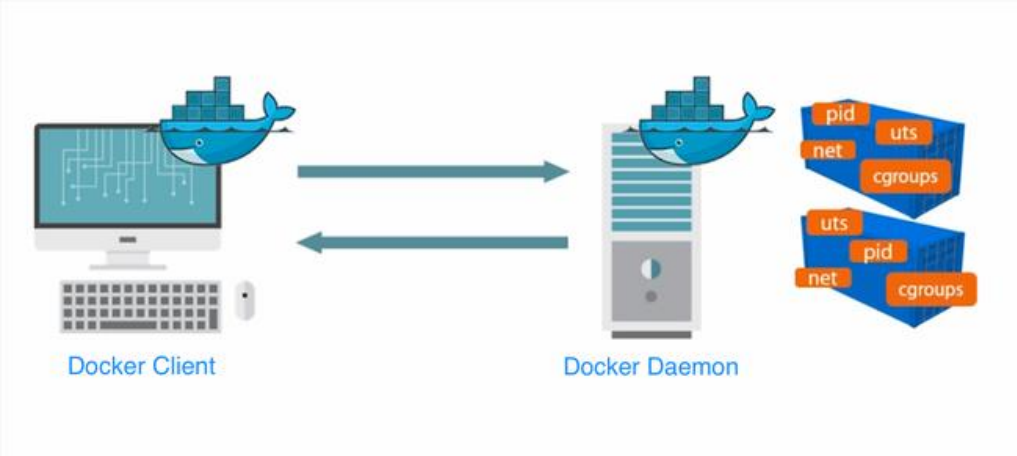

Figura 4 - Comunicação entre o *Docker Client* e *Docker Daemon [20]*

Quando a utilização de contêineres começa a ficar complexa, é necessária a utilização de orquestradores. Estes orquestradores coordenam a instanciação, escala, conexão, agendamento de tarefas e uma série de outras funções, abstraindo atividades complexas para o desenvolvedor ou gestor. Entre os orquestradores mais conhecidos estão *Kubernetes, Mesos, Swarm* e *Cattle.*

Os componentes principais da plataforma Docker [34], são:

- **Dockerfile**: É um arquivo de texto simples, composto por uma sequência de comandos, que funciona como uma “receita” para a criação de Docker Images.
- **Docker Images**: A base da criação de containers, uma imagem é composta da união de camadas de sistemas de arquivos, empilhadas umas sobre as outras, conforme a Figura 5.
- **Docker Registry**: São repositórios de imagens.

- **Docker Compose**: Possibilita a criação conjunta de grupos de containers para um propósito comum a partir de um script.

2.5 Sumário do Capítulo

Neste capítulo foi introduzido em qual contexto este trabalho está inserido, trazendo as problemáticas do desenvolvimento de produtos de TI, as metodologias solucionadoras dos problemas apontados por ela, suas práticas e modelos de ferramentas para auxílio da metodologia de desenvolvimento e operação dos produtos. No capítulo seguinte serão abordadas as ferramentas específicas que atuam em conjunto para o auxílio da metodologia DevOps, tais como as *ferramentas de provisionamento, gerenciadores de configurações* e *orquestradores de contêineres*.

3. Provisionamento, Gerenciamento de Configurações e Orquestramento de Contêineres

A cultura DevOps promove ganhos mútuos a uma equipe de TI, porém, são necessárias o uso de ferramentas que auxiliem em uma maior agilidade, eficiência e administração de recursos. Como parte fundamental deste processo, *ferramentas de provisionamento, gerenciadores de configurações* e *orquestradores de contêineres* são essenciais para automatizar várias etapas de um *deploy*, as quais serão estendidas ao longo deste capítulo.

3.1 Ferramenta de provisionamento

Ferramentas de provisionamento são responsáveis por criar os servidores de forma automatizada. Estas ferramentas podem fazer uso do provedor de nuvem, ou local, para criar uma gama de serviços, tais como banco de dados, firewalls, configurações de redes internas ou qualquer outro elemento de infraestrutura [21], de forma escalável, gerenciável e automática.

A conveniência e o menor investimento em tempo necessário para implantações de computadores em larga escala tornam as ferramentas de provisionamento uma ferramenta indispensável. Geralmente, os provisionadores fazem uso de *scripts,* com formatos de arquivos próprios, para seguirem certas instruções de trabalho, deixando os usuários personalizarem sua automação.

A ferramenta de provisionamento utilizada neste trabalho será o *Terraform*.

3.1.1 Terraform

Terraform é uma ferramenta para construção, mudança e versionamento de infraestrutura de forma rápida e segura [22]. Assim como o *docker-Compose*, o *Terraform* possui arquivos de configuração, nos quais descrevem os componentes que serão necessários para rodar uma única aplicação ou *cluster* por completo.

Os arquivos de configuração permitem que a ferramenta *Terraform* possa traçar um plano de execução, habilitando que sejam vistas as diferenças entre o

status atual da infraestrutura e o pretendido. Com as mudanças aferidas, a ferramenta fica disponível para a construção, destruição, incremento ou alteração da infraestrutura descrita nos arquivos de configuração.

Podemos assim, apontar as características do *Terraform*:

- A infraestrutura como código permite que a infraestrutura seja tratada em um alto nível e na forma de código, habilitando que esta seja usada mais de uma vez [22].
- O plano de execução mostra como a ferramenta irá atuar, evitando que problemas aconteçam durante a execução automática [22].
- O *Terraform* constrói recursos gráficos internos que indicam todos as dependências e não dependências de todos os seus recursos, permitindo que a infraestrutura seja construída mais eficientemente [22].

3.2 Ferramenta gerenciadora de configurações

As ferramentas que auxiliam na gestão de configurações possuem diversas funções. Elas normalmente controlam estados do sistema, ajudam a centralizar toda as configurações e facilitam a administração e criação de novos ambientes. Essas capacidades são significantes porque trazem consigo resolução de uma série de problemas na perspectiva do gerenciamento de sistemas, entre elas temos: “configuração de mudanças; controle e segurança; automação e aplicação; consistência no processo; proliferação da virtualização e da computação em nuvem” [23] (p.9). Uma ferramenta em destaque e bastante completa no mercado para auxiliar nesta tarefa é o *Rancher* [25], que foi selecionado para fazer parte deste trabalho.

3.2.1 Rancher

O *Rancher* é um software de código aberto com funções de gerenciamento de configurações dos contêineres *docker*. Esta plataforma gerenciadora de contêineres facilita o *deploy* e execução de contêineres em qualquer infraestrutura [24]. Por meio dele, é possível administrar todos os pontos de seu ambiente *Docker*, incluindo: orquestração, load balance, volumes, rede, configuração para registros privados e

públicos de imagens, além de fornecer facilidade aos usuários com os serviços de controle de usuários, *logs* de atividades e comunicação com os recursos do software via *API*.

O software consiste em basicamente quatro componentes principais, são eles: “ (1) orquestração de infraestrutura;(2) orquestração de contêiner e *scheduling* (agendamento) ;(3) catálogo de aplicações; e (4) grade de controle empresarial” [24]. Para fins deste trabalho, serão usados os componentes da orquestração de infraestrutura e orquestração de contêiner e *scheduling*. Mostra-se na Figura 5 todas as ferramentas que o *Rancher* administra e alguns exemplos de aplicações que são fornecidas pelo gerenciador para os usos de cada área.

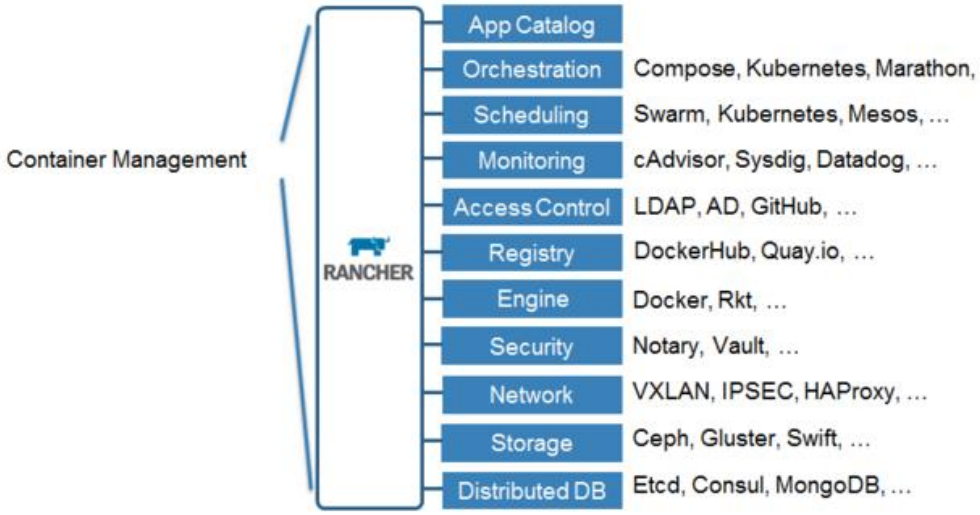

Figura 5 - Áreas de gerenciamento do *Rancher* [24]

3.2.1.1 Orquestração de Infraestrutura

O *Rancher* faz uso dos recursos computacionais brutos de qualquer nuvem pública ou privada na forma de servidores Linux, não fazendo distinção se a máquina hospedeira é física ou virtual. Ele apenas espera que a máquina o sirva com os recursos de CPU, memória, armazenamento em disco local e conectividade de rede.

Os serviços de infraestrutura fornecidos pelo *Rancher* incluem em Rede, armazenamento, balanceador de carga, DNS e segurança e que são executados como contêineres [25], podendo ser rodados em qualquer servidor com sistema operacional Linux.

3.2.1.2 Orquestração de contêiner e *scheduling*

Como resposta ao uso de diversos *frameworks* de orquestração e *scheduling,* a plataforma *Rancher* disponibiliza ao usuário o poder de escolha de uso dos orquestradores mais populares incluindo Mesos, Swarm e Kubernetes. Em adição a estes frameworks de orquestração e *scheduling*, o *Rancher* suporta o *Cattle*, um orquestrador originalmente projetado para ser uma extensão do Swarm, porém eles se divergiram e o *Cattle* continuou em desenvolvimento [24].

3.3 Ferramenta orquestradora de contêiner

As ferramentas de orquestração de contêiner têm diversas aplicações. Elas otimizam o processo operacional da infraestrutura de TI, abstraindo boa parte do contato com máquinas físicas e virtuais. Estas ferramentas possuem algumas características e objetivos em comum no desenvolvimento de orquestração, sendo elas a *Configuração Declarativa*, as *Regras e Restrições*, o *Provisionamento,* a *Descoberta de Serviços* e o *Monitoramento* [26].

A *Configuração Declarativa* provê ao usuário o poder do mesmo definir o estado em que deseja que a aplicação esteja através de configuração e a partir disso, o orquestrador trabalha na construção do estado e o mantém [26].

As aplicações geralmente possuem políticas ou requisitos especiais para manter o desempenho e a alta disponibilidade do servidor. Para isso, é necessário que *Regras e Restrições* sejam definidas pelo orquestrador para que haja uma melhor distribuição na carga de acesso por diferentes aplicações, deixando o sistema mais robusto e tolerante a picos de acesso [26].

Uma das principais características do orquestrador é fornecer um *Provisionamento* na distribuição dos contêineres dentro de diferentes hospedeiros e, baseado nas configurações do usuário, executar essa distribuição [26]. Além dessa característica, o orquestrador também precisa fornecer aos contêineres durante suas alocações, a *Descoberta de Serviços* desses contêineres por outros contêineres e vice-versa [26].

A configurações previamente ditadas pelos usuários devem ser *Monitoradas* e mantidas pelo orquestrador, no estado desejado. Caso um contêiner falhe, o orquestrador deve perceber e substituí-lo por outro, se um hospedeiro falhar, ele deve redistribuir os contêineres dele em outros [26].

Para este trabalho, será usado o *Cattle*, a principal ferramenta orquestradora de contêiner do *Rancher* [27], devido a sua simplicidade e maturidade em *deploys* de contêineres.

3.3.1 Cattle

Como falado anteriormente, o *Cattle* é uma ferramenta que provê ao usuário facilidades de uso. Isso devido a sua similaridade com o
Docker, já que seus comandos são baseados nos comandos *Docker* e também, o uso do *docker-compose* é para definir os planos da construção das aplicações [28].

Os *deploys* das aplicações são organizados em “pilhas”, que podem ser provisionados automaticamente a partir de arquivos de configuração do *docker-compose* e do *rancher-compose*. O *rancher-compose* é uma ferramenta que funciona similarmente ao do *docker-compose*, porém com funções de automatizar os processos de orquestração, como por exemplo os de escalamento e criação de contêineres, a partir das regras compostas em seu arquivo de configuração [29].

O *Cattle* é responsável pelo rastreamento de metadados, recursos, relacionamentos, estados, mudanças de estado e operações iniciais, tornando-o um orquestrador de alto nível [27].

3.4 Uso das Ferramentas para *deploys* de sistemas web

Na metodologia DevOps, é necessário que se tenha a utilização de ferramentas que automatizem o processo de *deploy*, as equipes sejam multidisciplinares, flexíveis e se modelem nas práticas da *integração*, *entrega* e *implantação contínua* [1].

A características das ferramentas apontadas auxiliam no processo de implementação de Sistemas Web, abrangendo as equipes de desenvolvimento e operação.

A ferramenta de provisionamento, *Terraform*, permite a construção de infraestrutura de forma automatizada através de um script de configuração [22], que possui as variáveis da infraestrutura a ser construída e da conexão com a *API* do *Rancher* [30].

O *Rancher*, por sua vez, controla todos os estados do orquestrador *Cattle*, ajudando a centralizar toda as configurações e facilitando a administração e criação de novos ambientes conteinerizados [23].

Todas estas ferramentas permitem que as equipes de desenvolvimento sejam mais independentes na criação de ambientes de *deploys* devido a facilidade na criação de infraestruturas virtualizadas e por scripts programáveis, além de facilitar a implementação das práticas da *integração*, *entrega* e *implantação contínua*.

A proposta deste trabalho é a de integrar os serviços do *Terraform* com o do *Rancher* utilizando-se a orquestração de contêineres com o *Cattle*, nomeando este projeto de *GenCloud*, e com isso, realizar o *deploy* de um sistema web no modelo do projeto *GenCloud* e em um modelo com os processos manuais. Esta abordagem será trazida no próximo capítulo.

4. Projeto GenCloud

O projeto GenCloud visa integrar algumas ferramentas com o intuito de agilizar no processo de *deploy* e aumentar a interoperabilidade entre as equipes de desenvolvimento e de operação, como falado no capítulo anterior. Este trabalho será realizado na Genomika Diagnósticos [31], utilizando-se seu servidor de alto desempenho, batizado de *Mendel*, e o sistema web para demonstração da aplicação, o *GenSoft,* o software mais importante da empresa. Para fins de análise, o GenCloud será implementado e comparado ao modelo atual de *deploy* da Genomika Diagnósticos, ambos com a finalidade de rodar a aplicação web (*GenSoft*), analisando índices relevantes que serão apontados ao longo deste capítulo.

4.1 Modelo atual

Para haver um melhor entendimento da criação do projeto GenCloud, é necessário a contextualização de como se encontra o modelo atual de *deploy* do *GenSoft,* na Genomika Diagnósticos.

O *GenSoft* é caracterizado por ser um sistema web direcionado para o contexto de análises clínicas. Ele automatiza vários processos internos da Genomika e é utilizado pela maior parte dos funcionários da empresa. Sua estrutura é composta pela linguagem *Python* [42], pelo gerenciador de tarefas *Celery* [43] e pelo servidor HTTP *Nginx* [44] e está continuamente em desenvolvimento.

O modelo atual de *deploy* se caracteriza pela presença de equipes de desenvolvimento e de operações, porém, cada uma delas com suas respectivas funções e sem um entendimento correto de como uma entende a outra.

A equipe de operações fica responsável por manter todo o servidor e a aplicação do *GenSoft* rodando, caso haja alguma falha nos serviços, ela imediatamente deve solucionar o problema. Toda a instalação de sistema operacional, preparação do ambiente virtual e instalação softwares necessários ao funcionamento da aplicação em produção também são de responsabilidade do time da infraestrutura. Todas essas operações são realizadas de forma manual e sem o

auxílio de ferramentas de automação, o que torna o processo cansativo e demorado e, possivelmente, sujeito a falhas.

A equipe de desenvolvimento é responsável por construir, atualizar e realizar o *deploy* do *GenSoft.* Ela aplica a metodologia Ágil em seu processo de desenvolvimento da aplicação, utilizando-se bastante do *Scrum,* onde a cada duas semanas, em média, termina um ciclo de produção. A cada ciclo, cada desenvolvedor fatora seu código, realiza testes unitários [40] e o integra ao código da aplicação através da ferramenta de gerenciamento de versão, o GitLab [41].

Com o código final pronto, o desenvolvedor entra remotamente no servidor via *SSH*, atualiza o código pelo GitLab, resolve os conflitos de código, se houver, e roda a aplicação. Assim como a equipe de operações, a equipe de desenvolvimento também realiza todo esse processo de forma manual, o que acarreta uma maior demora no processo de *deploy*, devido a minuciosidade necessária para minimizar os problemas de conflito de código e as possíveis novas instalações de novos softwares de apoio ao *GenSoft*, devido a instalação de novas bibliotecas. O maior prejudicado, na maioria das vezes, é o usuário final que precisa esperar o sistema ficar disponível, atrasando processos internos da empresa e podendo prejudicar o negócio.

4.2 GenCloud

O projeto GenCloud é uma solução de automatização os processos de *deploy* do modelo atual a partir da integração das ferramentas *Terraform*, *Rancher* e *Cattle*. Esta união permite o *deploy* de quaisquer tipos de sistemas web em nuvens públicas ou privadas com os conceitos de *conteinerização* e infraestrutura por código.

O valor que o GenCloud pretende passar é de que o desenvolvedor apenas se preocupe em usar dois comandos do *Terraform*, o '**terrafom plan**' para verificar as mudanças a serem feitas em relação à última configuração e o '**terraform apply**' para que o contêiner orquestrado pelo *Cattle* seja criado e siga um script de inicialização que seguirá uma série de passos contidos no Anexo A. O *script* contido no Anexo A contém todo o passo-a-passo que o desenvolvedor precisar realizar , caso esteja realizando o processo de *deploy* no modelo atual, falado anteriormente.

O *Terraform* pode ser executado de qualquer servidor ou computador que possua conexão com o servidor alocado para o *Rancher*, porém, para fins deste

trabalho, ele será rodado a partir da máquina remota para poder flexibilizar e facilitar o *deploy* do mesmo. Para o *Terraform* poder rodar, serão necessários dois arquivos de configuração, o do *docker-compose* contido no Anexo B, e o dele próprio, contido no Anexo C. A configuração contida no Anexo C contém informações de conexão com o *Rancher* e da “pilha” que será criada pelo *Cattle* com as configurações contidas no Anexo B, que contém as configurações dos contêineres que serão criados (*Gensoft*, *Postgres*, *Redis* e *Nginx*), as imagens que serão usadas, quais portas serão redirecionadas, as variáveis de ambiente que serão adicionadas e quais pastas serão montadas do servidor para o contêiner, além de estabelecer quais as relações entre os contêineres da “pilha”. Uma melhor projeção do GenCloud pode ser vista na Figura 6.

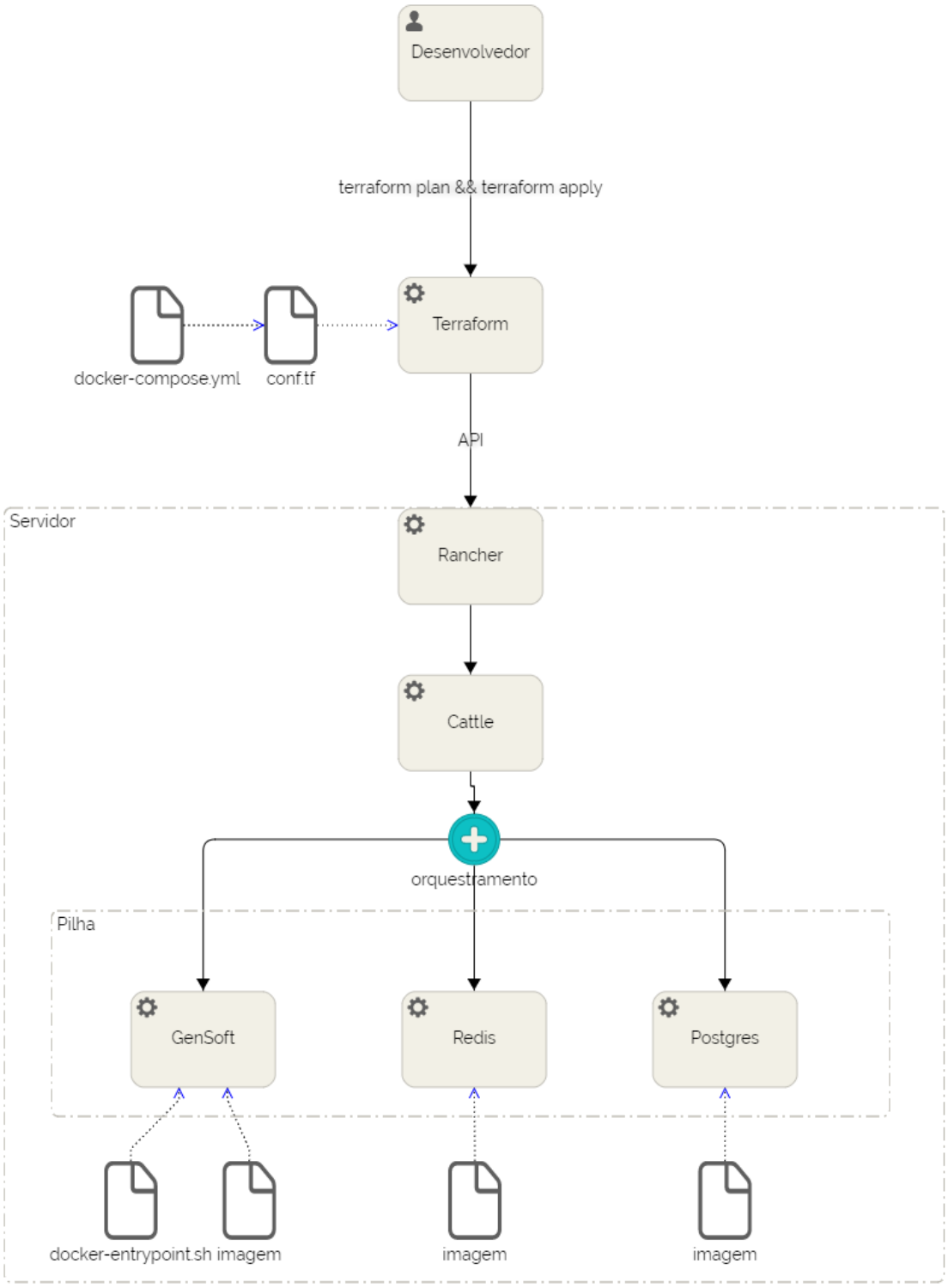

Figura 6 - Fluxo dos processos do GenCloud

Para haver mais rapidez na transferência e manter as imagens dos contêineres na empresa, foi criado um registro privado de imagens em um dos servidores da Genomika Diagnósticos. Isso permite que qualquer servidor ou computador com o *docker* instalado possa fazer uso de uma mesma imagem, como a que roda o

GenSoft, por exemplo. Isso habilita que os desenvolvedores desenvolvam com o mesmo ambiente de produção, porém, em suas próprias máquinas.

O projeto GenCloud permite que tarefas de infraestrutura de servidores, antes executadas pelo time de operações, sejam manuseadas pelos desenvolvedores devido a facilidade na criação da infraestrutura por linhas de comandos. A equipe de operações apenas fica responsável por prover e manter os serviços falados anteriormente.

4.3 Técnica e métricas

Para ser realizada a comparação entre os modelos relatados anteriormente será utilizada a técnica de simulação. Esta técnica foi escolhida por utilizar programação para fazer a interação e medição das métricas escolhidas e ter um custo e precisão moderados, se comparados com uma modelagem analítica ou uma medição [32, p. 44]. Ao analisar métricas de performance, pode-se olhar para velocidade, confiabilidade e disponibilidade [32, p. 48]. Para esta avaliação, serão utilizadas uma métrica para velocidade e uma para disponibilidade:

- *Velocidade da criação do ambiente virtual* significa mensurar o tempo necessário para instalar todos os pacotes de software necessários para o *GenSoft* no ambiente virtual do servidor.
- *Disponibilidade do sistema GenSoft* para o usuário significa o tempo levado para o software entrar em operabilidade, sendo em qualquer nível de aplicação (produção, testes, homologação ou qualquer outro).

Como serão comparados diferentes modelos, é relevante analisar os processos de *deploy* de cada um, a fim de entregar ao usuário, ou usuários, opções de escolha que se encaixe em seu perfil de desenvolvimento.

4.4 Configurações da Infraestrutura

Para realizar as comparações a seguir, serão utilizadas duas máquinas virtualizadas no servidor *Mendel*, com todos os recursos computacionais fornecidos pela Genomika Diagnósticos. Cada máquina será virtualizada pelo XenServer [33], ferramenta

opensource de orquestramento de máquinas virtuais, com as mesmas especificações. A Tabela 2 mostra as configurações de cada umas dessas máquinas.

Máquina Virtual	Memória	vCPU	Armazenamento	Sistema Operacional
MV 1	8 GB	4	350 GB	Ubuntu-Server 14.04
MV 2	8 GB	4	350 GB	Ubuntu-Server 14.04

Tabela 2 - Recursos alocados para cada máquina virtual do servidor *Mendel*

Cada máquina virtual rodará com o sistema operacional *Ubuntu Server 14.04,* isso devido a este ser uma versão mais enxuta para servidores da distribuição *Ubuntu* e possuir compatibilidade com o *Docker v1.12.6 Rancher v1.6,* todas ferramentas usadas no GenCloud, além de já vir nativo o *python v3.4,* na qual é uma linguagem de programação necessária para rodar o *GenSoft*. Sendo assim, uma única distribuição de sistema operacional supre as necessidades da implementação dos dois modelos a serem comparados. O *Terraform v0.9.8* será instalado em um computador remoto, para execução comandos.

Para o *GenSoft*, no modelo do GenCloud, haverá uma imagem nomeada de **dockerregistry.genomika.com/gensoft:base,** cujo *Dockerfile* se encontra no Anexo D e na qual já tem instalado todos os softwares necessários para a aplicação rodar, até a entrega deste trabalho. Esta imagem foi construída a partir de uma outra imagem nomeada de **dockerregistry.genomika.com/genomika-ubuntu:14.04**, com o Dockerfile na qual foi construída com base na imagem do *ubuntu:14.04* e tem instalado todos os softwares básicos para as outras aplicações da Genomika Diagnósticos, como por exemplo, o *SSH* e o *GIT*.

Considerando-se que cada modelo tem seus modos de execução do *GenSoft,* serão elencados alguns requisitos para se estabelecer uma base de comparação.

4.5 Análise

Para se obter os resultados dos testes de performance, serão analisadas duas métricas em ambos os modelos, a da *velocidade da criação do ambiente virtual* e da *disponibilidade do sistema GenSoft.*

O teste da *velocidade da criação do ambiente virtual* visa mensurar o tempo de instalação de todos os softwares de dependência do *GenSoft*. Para realizar este processo no modelo atual de *deploy*, será instalado manualmente na máquina virtual *MV 1* os softwares do *Postgres*, *Redis* e *Nginx* apresentados no *docker-compose* do Anexo B e os softwares apresentados nos *Dockerfiles* do Anexo D e E, onde cada comando *RUN* representa uma linha a ser inserida no terminal do servidor. No modelo GenCloud, os *Dockerfiles* serão rodados na máquina virtual *MV2* através dos comandos '**docker build -t "dockerregistry.genomika.com/ubuntu-genomika:14.04" .**' para o Anexo E, e o '**docker build -t "dockerregistry.genomika.com/gensoft:base**' **.**' para o Anexo D, sequencialmente, devido ao nível de dependência entre elas. Cada comando terá um tempo de execução e a soma deles representará o tempo final para o teste.

O segundo teste, o da *disponibilidade do sistema GenSoft*, visa mensurar o tempo para disponibilizar o *GenSoft* na rede e a nível de produção. No modelo atual de *deploy* este processo será realizado de forma manual, executando os mesmos comandos disponibilizados no Anexo A. No modelo do GenCloud, como falado anteriormente, serão executados os comandos '**terraform plan**' e o '**terraform apply**' sequencialmente de um computador remoto. Cada comando terá um tempo de execução e a soma deles representará o tempo final do teste.

4.6 Resultados

Os testes apontados anteriormente apresentam resultados bastante significativos. Eles trazem números que representam uma maior eficiência no desenvolvimento de sistemas web, sendo válido tirar-se conclusões de cada etapa.

As métricas apontadas na Tabela 3, referente à *velocidade da criação do ambiente virtual*, mostram que na *MV 1*, precisou-se de menos de cinquenta minutos para preparar todo o ambiente de trabalho, em relação a *MV 2*, por mais que nesta última tenha sido necessário rodar dois comandos para criar duas imagens de contêiner. A instalação de alguns softwares na *MV 1* foram mais demoradas devido

aos trabalhos manuais e a necessidade de configurar algumas variáveis, com o *Postgres* como exemplo, e enquanto na *MV 2* todas as configurações já estavam pré-configuradas.

Máquina Virtual	Tempo de criação
MV 1	1h e 30 min
MV 2	1º comando (11m) + 2º comando (31min)

Tabela 3 - Resultados do teste da *velocidade da criação do ambiente virtual*

Na Tabela 4 foram medidos os tempos necessários para a *disponibilidade do sistema GenSoft*. As diferenças entre os tempos de execução em ambas as máquinas aparentemente não são grandes, em comparação ao teste anterior, porém, na *MV 1* todo o processo é manual, falhas de *deploy* podem acontecer e o tempo de execução pode aumentar.

Máquina Virtual	Tempo de execução
MV 1	180s (3 min)
MV 2	120s (2min)

Tabela 4 - Resultados do teste de *disponibilidade do sistema GenSoft*

4.7 Considerações

De acordo com os resultados obtidos, é notável a diferença de performance entre o modelo atual de deploy da Genomika Diagnósticos e o modelo do projeto GenCloud, sendo este último uma solução muito mais ágil e eficiente. Esta abordagem, em relação à solução antiga, possibilita:

- Automação da maior parte dos processos de *deploy* de sistemas web, incluindo o *GenSoft*.
- Menor *downtime* dos sistemas.

- O encapsulamento das aplicações e a mobilidade entre ambientes computacionais distintos.
- Relacionamento mais “saudável” entre as equipes de operações e de desenvolvimento.

Os resultados significativos proveem a equipe de TI com uma maior eficiência na entrega do produto, por mais que o esforço inicial na aplicação do GenCloud seja grande. Com mais tempo disponível, o time pode alocar mais tempo a outras atividades importantes.

5. Conclusão

Com o advento de novas metodologias de gestão de equipes de TI, a fim de aumentar a eficiência da produção de softwares e melhorar o relacionamento entre as diferentes áreas do time, novas tecnologias surgiram para apoiar os novos processos de desenvolvimento. A adoção de tecnologias modernas de containers de software em ambientes de *deploys* em nuvens, sejam elas públicas ou privadas, conforme este trabalho buscou mostrar, pode contribuir na solução de vários problemas de desenvolvimento de software.

Este trabalho conseguiu apresentar uma comparação entre métricas de um modelo manual, ainda bastante típico, de *deploy* e uma solução de modelo de *deploy* baseada em novas tecnologias de virtualização de ambientes. Os resultados são significativos devido aos níveis de automação, flexibilização e escalabilidade que o projeto GenCloud consegue apresentar, características que o outro modelo comparado não apresenta. Certamente, a proposta oferecida pelo GenCloud é uma solução que requer muitas mudanças e aprendizado a todos os membros da equipe de TI, sendo necessária uma análise profunda do gestor para saber se é viável ou não a aquisição da proposta.

Algumas das principais dificuldades encontradas na realização deste trabalho foram devidas ao aprendizado das ferramentas e a implementação do GenSoft e os softwares de dependência, de forma automatizada. A primeira se deve ao baixo nível de detalhes e organização na documentação das ferramentas e a segunda devido à grande carga de falhas aos vários testes.

Trabalhos futuros poderão ser realizados com o uso de mais ferramentas do *Rancher*, como o *rancher-compose*, que oferece recursos personalizados para o orquestramento dos contêineres, abrindo portas para a disponibilização de mais performance. Outro ponto que poderia ser melhorado seria em um melhor nível de automatização, com uma reorganização no *docker-compose* e no *entrypoint*.

Referências

1 Jonker, Margot. "DevOps Implementation Model for Large IT Service Organizations." (2017)

2 Beck, K. et al. (2001). Manifesto for Agile Software Development. Agile Alliance

3 Dingsoyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. Journal of Systems and Software, 85(6), 1213–1221. Disponível em: <http://doi.org/10.1016/j.jss.2012.02.033>

4 Yu, X., & Petter, S. (2014). Understanding agile software development practices using shared mental models theory. Information and Software Technology, 56(8), 911–921. Disponível em: <http://doi.org/10.1016/j.infsof.2014.02.010>

5 Schwaber, K. (1995). Scrum development process. Proceedings of the Workshop on Business …, (April 1987), 10–19. Disponível em: <http://doi.org/10.1007/978-1-4471-0947-1_11>

6 Schwaber, K., & Beedle, M. (2001). Agile Software Development with Scrum. cdswebcernch (Vol. 18). Disponível em: <http://doi.org/10.1109/2.947100>

7 Barlow, J., Giboney, J. S., Keith, M. J., Wilson, D. W., Schuetzler, R. M., Lowry, P. B., & Vance, A. (2011). Overview and Guidance on Agile Development in Large Organizations. Communications of the Association for Information Systems, 29, 25–44. Disponível em: <http://doi.org/10.2139/ssrn.1909431>

8 Fitzgerald, B., & Hartnett, G. (2005). A Study of the Use of Agile Methods within Intel. Business Agility and Information Technology Diffusion, 187–202. Disponível em: <http://doi.org/10.1007/0-387-25590-7_12>

9 Tessem, B. ., & Iden, J. . (2008). Cooperation between developers and operations in software engineering projects. Proceedings - International Conference on Software Engineering, (January 2008), 105–108. Disponível em: <http://doi.org/10.1145/1370114.1370141>

10 Walls, M. (2013). Building a DevOps Culture. O'Reilly Media, Inc.

11 Kim, G., Behr, K., & Spafford, G. (2013). The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win, 345.

12 Just Digital [online]. Disponível em: <http://blog.justdigital.com.br/devops-qual-a-diferencas-entre-continuous-delivery-continuous-integration-e-continuous-deployment/>

13 MOUAT, A. Using Docker: Developing and deploying software with containers.

14 Aprenda Cloud [online]. Disponível em: <http://www.aprendacloud.com.br/containers-docker-vms/>

15 RUBENS, P. What are containers and why do you need them? 2015. Disponível em: <http://www.cio.com/article/2924995/software/what-are-containers-and-why-do-you-need-them.html>.

16 Aprenda Cloud [online]. Disponível em: <http://www.aprendacloud.com.br/containers-docker-vms/>.

17 Klauslaube [online]. Disponível em: <http://klauslaube.com.br/2017/01/14/uma-ode-ao-docker-parte-1.html>.

18 LINTHICUM, D. The essential guide to software containers in application development. 2016.

19 Docker Docs [online]. Disponível em: <https://docs.docker.com/engine/docker-overview/#what-can-i-use-docker-for>.

20 svbtle [online]. Disponível em: <https://svbtleusercontent.com/q58xgz8o0dzgyg_small.png>.

21 Yevgeniy,B (2017). Terraform: Up and Running: Writing Infrastructure as Code [S.l.]: O'Reilly Media, 2015. ISBN 9781491915899.

22 Terraform [online]. Disponível em: <https://www.terraform.io/intro/index.html>.

23 Meyler, K.,Holt, B., Oh, Sandys, J. Oh, M., Ramsey, G., (2013). System Center 2012 Configuration Manager (SCCM) Unleashed.

24 Disponível em: <http://rancher.com/announcing-rancher-1-0-ga/>.

25 Rancher [online]. Disponível em: <http://rancher.com/rancher>.

26 20 MSV, J. From containers to container orchestration. 2016. Disponível em https://thenewstack.io/containers-container-orchestration/: .

27 GitHub [online]. Disponível em: <https://github.com/rancher/cattle>.

28 Rancher [online]. Disponível em: <http://rancher.com/cattle-swarm-kubernetes-side-side/>.

29 Rancher Docs [online]. Disponível em: <http://docs.rancher.com/rancher/v1.6/en/cattle/rancher-compose/>.

30 Terraform [online]. Disponível em: <https://www.terraform.io/docs/providers/rancher/index.html>.

31 Genomika Diagnósticos [online]. Disponível em: <https://www.genomika.com.br/>.

32 JAIN, R. The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling. New York: Wiley, John & Sons, 1991. ISBN 9780471503361.

33 XenServer [online]. Disponível em: <https://xenserver.org>

34 JERNIGAN, T. Docker 1.11 et plus: Engine is now built on runC and containerd. 2016. Disponível em: <https://medium.com/@tiffanyfayj/docker-1-11-et-plus-engine-is-now-built-on-runc-and-containerd-a6d06d7e80ef>.

35 Disponível em: <https://4.bp.blogspot.com/-L0bNvBeUJ0M/VwAwnxFmZEI/AAAAAAAABcg/kitQ5yqe3BsIprPs_E714JzVjIRPVN72w/s1600/CI_CDY_CDT_04.png>.

36 CLUSTERHQ. Container Market Adoption Survey 2016. Disponível em: <https://clusterhq.com/assets/pdfs/state-of-container-usage-june-2016.pdf> .

37 DOCKER INC. What is Docker? 2016. Disponível em: <https://www.docker.com/what-docker>.

38 RAJKUMAR, M. et al. Devops culture and its impact on cloud delivery and software development. 2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Spring), Institute of Electrical and Electronics Engineers (IEEE), 04 2016.

39 TeraData [online]. Disponível em: <http://blogs.teradata.com/data-points/wp-content/uploads/2016/03/Devops-image-March-2.png>.

40 Gilson, W. Ferraz, A. TESTANDO MAIS PARA DEPURAR MENOS: OS BENEFICIOS DOS TESTES PARA O DESENVOLVIMENTO DE SOFTWARE (2014). Disponível em: <http://intertemas.toledoprudente.edu.br/revista/index.php/ETIC/article/viewFile/4118/3878>

41 GitLab [online]. Disponível em: <https://about.gitlab.com/features/>

42 Python [online]. Disponível em: <https://docs.python.org/3/>

43 Celery [online]. Disponível em: <http://www.celeryproject.org/>

44 Nginx [online]. Disponível em: <https://www.nginx.com/resources/wiki/>

45 GOOGLE TRENDS. Docker. 2016. Disponível em: <https://trends.google.com.br/trends/explore?q=\%2Fm\%2F0wkcjgj> .

46 KANG, H.; LE, M.; TAO, S. Container and microservice driven design for cloud infrastructure devops. 2016 IEEE International Conference on Cloud Engineering (IC2E), Institute of Electrical and Electronics Engineers (IEEE), 04 2016.

ANEXO A -- Script docker-entrypoint.sh

```
#!/bin/bash
/usr/sbin/sshd -D &!

git clone git@gitlab.com:danilo2/gensoft.git
git clone git@gitlab.com:genomika/illuminate.git

cd gensoft
git checkout $gensoft_branch
pip3 install -r requirements/base.txt
pip3 install -r requirements/dev.txt

cd ..
cd illuminate
git checkout $illuminate_branch
python3 setup.py install



cp /var/www/gensoft/.env_homo /var/www/gensoft/.env
cd /var/www/gensoft

echo "yes" | invoke collect

if [ "$migrate" = true ] ; then
        echo "yes" | invoke mig
fi
celery -A genomika_soft worker --loglevel=info &!
celery -A genomika_soft worker --loglevel=info --beat &!
celery flower --broker=redis://$redis_ip:$redis_port --persistent=true &!
gunicorn --workers=5 --bind 0.0.0.0:$port --timeout 600 --limit-request-line 0 --
access-logfile /media/gunicorn-access.log --error-logfile /media/gunicorn-error.log -
-settings=genomika_soft.settings.$gensoft_settings
genomika_soft.wsgi:application
```

ANEXO B -- docker-compose.yml

```
version: '2'
services:
  gensoft:
    image: dockerregistry.genomika.com/gensoft:base
    entrypoint:
    - /media/entrypoint/docker-entrypoint.sh
    environment:
    - gensoft_branch=master
    - illuminate_branch=master
    - migrate=true
    - port=8000
    - PYTHONIOENCODING=UTF-8
    - C_FORCE_ROOT="yes"
    - redis_ip=172.16.225.6
    - redis_port=6379
    volumes:
    - /home/rancher-apps/apps/gensoft/entrypoint/gensoft/:/media/entrypoint/
    - /home/rancher-apps/apps/gensoft/env/:/media/env/
    - /home/rancher-apps/apps/gensoft/celerylog/:/media/celerylog/
    - /home/rancher-apps-homolo/apps/gensoft/mediafiles/:/gensoft/mediafiles/
    - /home/rancher-apps-homolog/apps/gensoft/static/:/gensoft/static/
    ports:
    - 8000:8000/tcp
    - 2222:22/tcp
    - 5555:5555
    depends_on:
    - redis
    - nginx
    links:
    - redis
  nginx:
    image: nginx:latest
    volumes:
    - /home/rancher-apps/apps/gensoft/nginx/:/etc/nginx/conf.d
    - /home/rancher-apps/apps/gensoft/static/:/static/
    - /home/rancher-apps/apps/gensoft/mediafiles/:/mediafiles/
    ports:
    - 80:80/tcp
  redis:
    image: redis:latest
    container_name: rd01
    ports:
    - '6379:6379'
    links:
    - gensoft
  db:
    image: sameersbn/postgresql:9.6-2
    environment:
      DB_PASS: *
```

```
    DB_USER: genomika
    DEBUG: 'false'
  volumes:
  - /home/rancher-apps/apps/db-homolog/db/:/var/lib/postgresql
  ports:
  - 5433:5432/tcp
```

ANEXO C -- Terraform conf.tf

```
# Configure the Rancher provider
provider "rancher" {
  api_url    = "http://172.16.225.6:8080"
  access_key = "47C89343774D6CDBA906"
  secret_key = "H7tMa3dTEF4PbrwxZbTd8mQKDMHW32bvMkCfQTG5"
}

resource rancher_host "foo" {
  name           = "rancher"
  environment_id = "1h4"
  hostname       = "rancher"
}

resource "rancher_stack" "external-dns" {
  name           = "gensoft"
  environment_id = "1a5"
  docker_compose = "${file("docker-compose.yml")}"
  start_on_create = "True"
}
```

ANEXO D -- Dockerfile

dockerregistry.genomika.com/gensoft:base

```
FROM dockerregistry.genomika.com/ubuntu-genomika:14.04

#libs
RUN apt-get install pdftk -y
RUN apt-get install unixodbc-dev -y
RUN apt-get install libmysqlclient-dev -y
RUN apt install libpq-dev python3-dev -y
RUN apt-get -y install libncurses-dev
RUN apt-get install libffi-dev -y
RUN apt-get install libssl-dev -y
RUN apt-get install python3-dev -y
RUN apt-get install python-dev -y
RUN pip3 install Cython
RUN pip3 install gunicorn==19.1.1
RUN pip3 install django-celery
RUN pip3 install pycodestyle
RUN pip3 install markupsafe
RUN sudo apt-get install libpulse-dev libjpeg-dev -y
RUN apt-get install libxml2-dev libxslt1-dev -y
RUN pip3 install redis

#commands
WORKDIR /var/www
RUN git clone git@gitlab.com:genomika/gensoft.git
RUN git clone git@gitlab.com:genomika/illuminate.git
RUN pip3 install -r gensoft/requirements/base.txt
RUN pip3 install -r gensoft/requirements/dev.txt
RUN python3 illuminate/setup.py install
RUN rm -rf illuminate
RUN rm -rf gensoft
```

ANEXO E -- Dockerfile dockerregistry.genomika.com/ubuntu-genomika:14.04

```
FROM ubuntu:14.04


RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:g3n0m1k@' | chpasswd
RUN sed -i 's/PermitRootLogin without-password/PermitRootLogin yes/'
/etc/ssh/sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional
pam_loginuid.so@g' -i /etc/pam.d/sshd

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

RUN apt-get install git-all -y
RUN apt-get install python3-pip -y
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```